Getting Started with tinyAVR® 2 Family ADC Hands-On

Prerequisites

- **Hardware Prerequisites**
  - The Microchip ATtiny1627 Curiosity Nano Evaluation Kit www.microchip.com/developmenttools/ProductDetails/DM080104
  - The Microchip Curiosity Nano Base for Click boards™ www.microchip.com/DevelopmentTools/ProductDetails/AC164162
  - The Force Click board www.mikroe.com/force-click
    - **Note:** The Force Click board is used for demo purposes, and any analog sensor can be used. Result graphs may vary from setup to setup.
  - Micro-USB cable (Type-A/Micro-B)
  - Two through-hole resistors, one 100 kΩ and one 33 kΩ
- **Software Prerequisites**
  - Microchip Studio 7.0.2397 or later
  - Microchip Studio ATtiny_DFP version 1.4.308 or above
  - MPLAB® Data Visualizer Stand-alone version 1.1.793 or above
- **Other**
  - The Microchip Studio Solutions projects available in

- **Estimated Completion Time: 210 minutes**

Introduction

Embedded applications often rely on analog inputs. Most Microchip MCUs are equipped with an integrated Analog-to-Digital Converter that can be used to gather data from the physical world. A baseline ADC will be sufficient to gather data under ideal conditions, but end-user operating environments are often far from ideal. Noisy on-board circuitry, noisy environments, and noisy power supplies are common examples of complicating factors. Conventional ADCs will often require hardware circuitry and firmware routines dedicated to signal conditioning, but Microchip's latest ADCs remove some of this overhead through advanced Core Independent functionality.

The new 12-bit ADC offers true differential measuring capabilities with optional hardware accumulation of up to 1024 samples, which effectively gives up to 17 bits resolution. With sampling rates up to 375 ksps, the analog data is rapidly available. The Programmable Gain Amplifier (PGA) for the ADC can amplify single-ended and differential analog inputs to make it possible to measure even small amplitude signals efficiently.

This hands-on training covers the following topics:

- Assignment 1: Single-Ended ADC
- Assignment 2: Understanding ADC Modes
- Assignment 3: 17-bit Resolution
- Assignment 4: Read Temperature
- Assignment 5: Differential ADC

- Assignment 6: Using Programmable Gain Amplifier (PGA)

This training manual goes through assignments that give a general introduction to the ADC module. Some of the assignments are more specific to the 12-bit ADC and demonstrates the added features such as the PGA. Basic theory related to each assignment is introduced at the beginning of each assignment, then the functionality is verified through the MPLAB *Data Visualizer*. Every assignment is provided with a preconfigured code, where most code is already in place, and only essential modules for the specific assignment need to be configured.

Table of Contents

1. Icon Key Identifiers

The following icons are used in this document to identify different assignment sections and to reduce complexity.

Info: Delivers contextual information about a specific topic.

Tip: Highlights useful tips and techniques.

To do: Highlights objectives to be completed.

Result: Highlights the expected result of an assignment step.

Indicates important information.

Execute: Highlights actions to be executed out of the target when necessary.

2. Assignment 1: Single-Ended ADC

In this assignment, the ATtiny1627 will be configured to stream samples to the MPLAB *Data Visualizer*.

The ADC will be in Single-Ended mode, and two operational modes will be demonstrated. The two modes are called Single mode and Burst Accumulation mode.

Configure the Force Click board to generate an analog input signal for the ATtiny1627 ADC. This input signal will be sampled, and the results will be transmitted over USART to the *Data Visualizer*. Configure the *Data Visualizer* to graph the received data for visualization.

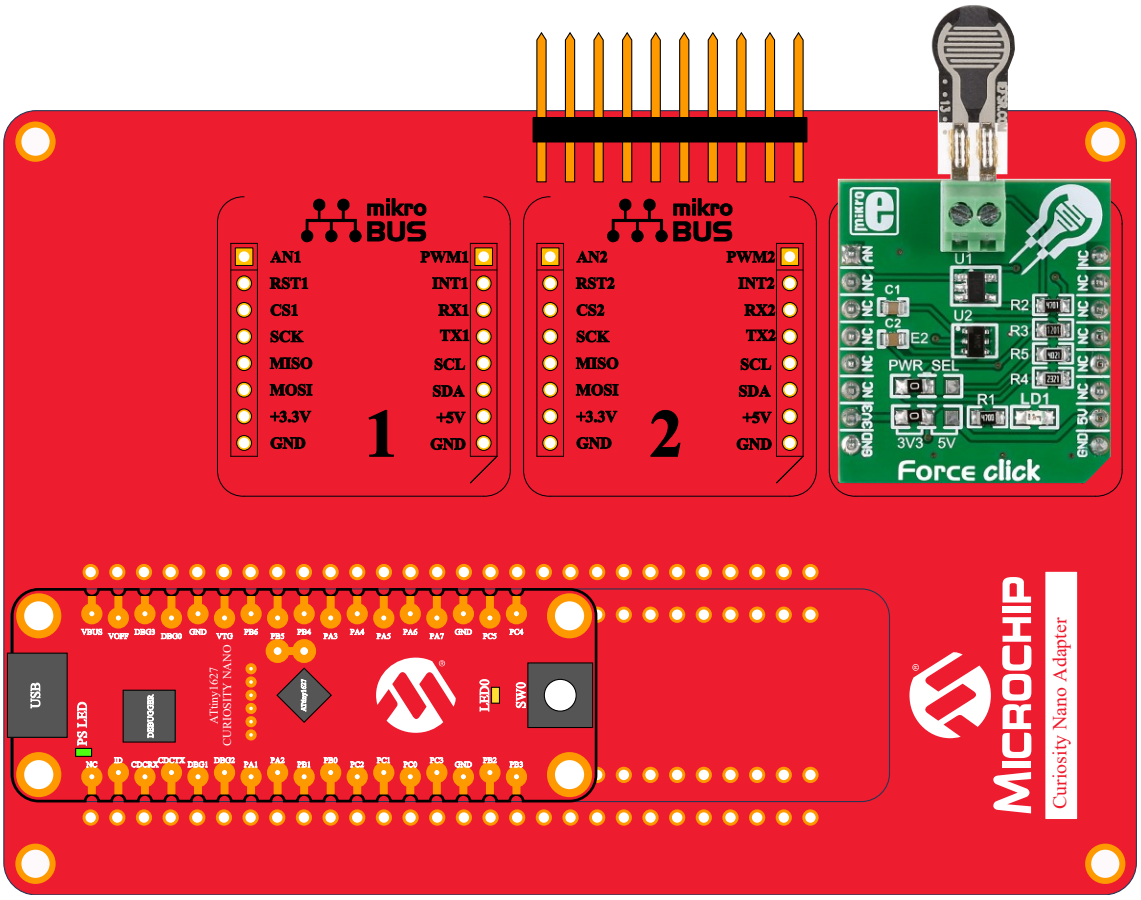
The Microchip Studio Solution *Assignment1* includes the required starting point for this assignment and should be available in the training material folder at the following path: `Assignment1`.

- **Objectives**
  - Get familiar with the setup
  - Open the assignment project in Microchip Studio
  - Get familiar with the assignment project
  - Configure the ADC
  - Use *Data Visualizer* to graph received ADC results

2.1 Hardware Setup

To do: Connect the Force Click board and the ATtiny1627 to the Curiosity Nano Base for Click boards, as shown in Figure 2-1.

Figure 2-1. Assignment 1: Hardware Setup



Info: The Force Click board is designed to use either a 3.3V or 5V power supply. The Force Click board in these assignments is powered by 3.3V. The on-chip LD1 will light up when the device is powered. For more info about the Force Click board, go to www.mikroe.com/force-click.

Info: The Curiosity Nano Base for Click boards™ is a hardware extension platform to ease the connection between Curiosity Nano kits and extension boards like the mikroBUS™ Click modules. For more information, go to www.microchip.com/Developmenttools/ProductDetails/AC164162.

1. Connect the ATtiny1627 Curiosity Nano Board to the socket of the Curiosity Nano Base.
2. Connect the Force Click board to mikroBUS™ socket number 3.

2.1.1 Hardware Walk-Through

The Force Click board from Mikroe is a mikroBUS™ add-on board that adds a force-sensing resistor to the project. The force-sensing resistor is a thin sensor consisting of two membranes separated by a spacer around the edges.

When pressed, the gap between the two membranes gets closed, which shorts the two membranes together with a resistance proportional to the applied force.

The force-sensing resistor is mounted at the front of the board and can be swapped with other force-sensing resistors if needed. The force sensitivity range is from about 0.2N to 20N. The measurements from the Force Click board is output on the AN pin as an analog voltage. The voltage ranges from GND to the supplied voltage. The board is simple to use and does not require any drivers. It communicates with the MCU via the mikroBUS™ AN pin.

2.2 Get Familiar with the Assignment Project

To do: Get familiar with the hands-on project, and verify that the project builds with no errors.

1. Open the assignment project *Assignment1* by following the steps listed below:
   1.1. Open *Microchip Studio*.
   1.2. Select *File* → *Open* → *Project/Solution...*
   1.3. Navigate to *Assignment1* and open *Assignment1.atsln*.

Info: The hands-on project will be loaded, and the project content listed in the *Solution Explorer* panel. If required, open the solution explorer by selecting *View* → *Solution Explorer.*

2. Study the assignment project structure. Open the .c files listed in the *Solution Explorer* and study the implementation of various functions called from *main.c*.

Info: The *main.c* should appear as shown below:

```
#define F_CPU 10000000ul

#include <avr/io.h>
#include <math.h>
#include <stdbool.h>
#include <util/delay.h>
#include <stdlib.h>

#include "main.h"
#include "clock_config.h"
#include "io.h"
#include "usart.h"
#include "adc.h"
#include "data_streamer.h"

int main(void)
{
    clock_init();
    io_init();
    adc_init();
    usart0_init();

    while (1)
    {

    }
}
```

Info:
- `clock_init()`, *clock._config.c*: Initializes the CPU clock to 10 MHz in *clock_config.c*.
- `io_init()`, *io.c*: Configures the directions of IO pins used in the hands-on application:
  - Pin PB2, output, TXD
  - Pin PB7, output, LED
  - Pin PA5, analog input, ADC
- `usart0_init()`, *usart.c*: Initializes the USART:
  - Baud rate: 115200
- `adc_init()`, *adc.c* is empty and will be implemented as a part of this assignment

Tip: To see how and where functions and definitions are implemented, *right-click* a function followed by *goto implementation*.

3. Build the project by selecting *Build → Build Solution* or by pressing F7. This will generate the project dependencies list.

Info: `avr/io.h` is included at the top of *main.c*. This header file is common for all AVR® devices, and including this file will ensure that the appropriate device header file is included, based on selected project settings. Open *iotn1627.h* from *Solution Explorer → Dependencies* to view the ATtiny1627 device header file. Additional information on coding style and structure is available in the AVR1000 application note at www.microchip.com/wwwappnotes/appnotes.aspx?appnote=en591581.

4. Make sure the project was successfully built in the previous step.

Tip: If there were build errors, the error list should appear automatically, but it can also be opened manually by selecting *View → Error List* from the top menu bar.

Result: The project is initialized and ready for development.

2.3 ADC Single Mode

The ADC can be configured in different operation modes. These modes can be split into three groups:

- Single mode - Single conversion per trigger, with 8- or 12-bit conversion output
- Series Accumulation mode - One conversion per trigger, with an accumulation of n samples
- Burst Accumulation mode - A burst with n samples accumulated as fast as possible after a single trigger

Series and Burst modes utilize 12-bit conversions and can be configured with or without scaling the accumulated result.

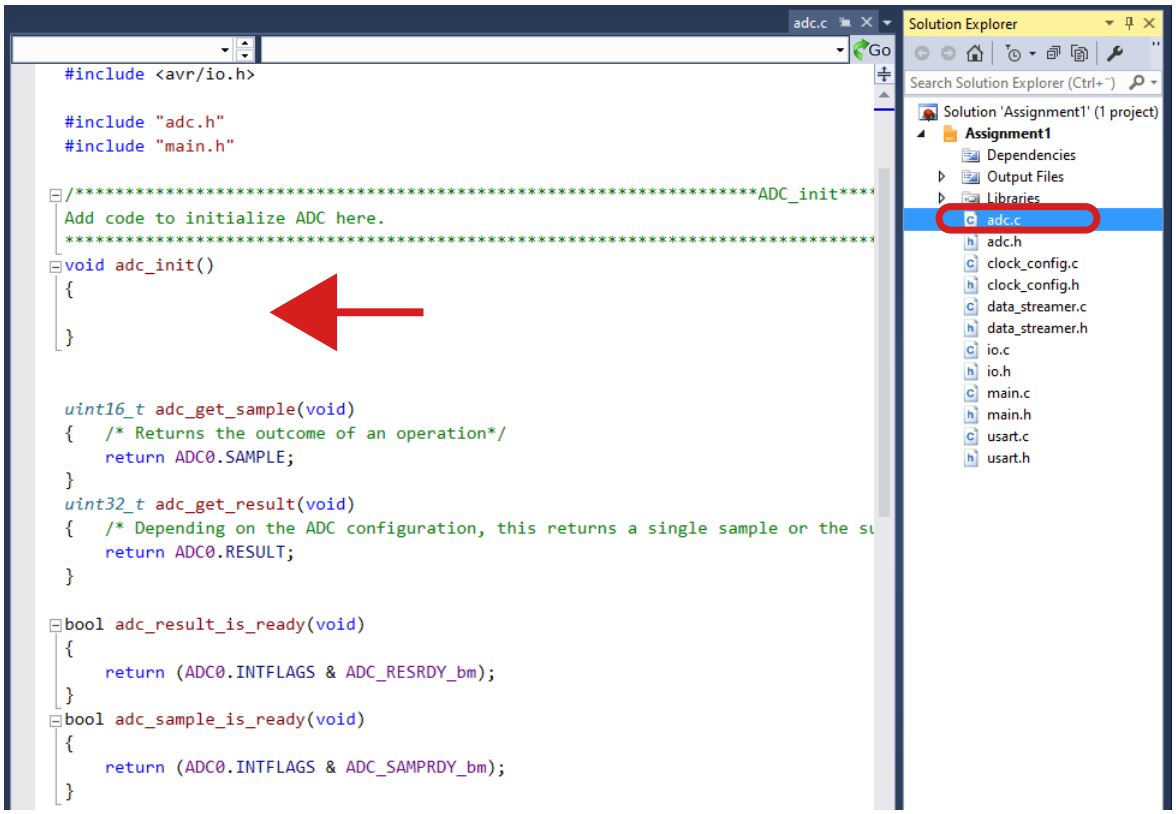
In this assignment, the focus is on Single mode and Burst Accumulation mode. Configuration of both setups will be shown starting with Single mode.

To do: Edit `adc_init()` to configure the ADC as listed below:

- ADC reference voltage: V_{DD}
- ADC Clock: CLK_PER/4
- Input Channel: AIN5 (Pin PA5)
- ADC sampling in Single mode
- Resolution: 12 bits

1. Open *adc.c* by double clicking *adc.c* in the *Solution Explorer*, and locate `adc_init()`. Figure 2-2 illustrates the expected result.

Figure 2-2. Assignment 1: adc_init()



2. Enable the ADC by writing a '1' to the ENABLE bit in the Control A (ADCn.CTRLA) register:

```
ADC0.CTRLA = ADC_ENABLE_bm;
```

3. Configure the ADC prescaler to produce the desired CLK_PER/4 ADC clock by writing the Control B (ADCn.CTRLB) register:

```
ADC0.CTRLB = ADC_PRESC_DIV4_gc;
```

Tip: Refer to the ATtiny1627 device data sheet and family manual available from www.microchip.com/wwwproducts/en/ATTINY1627 for detailed register descriptions.

4. Select the reference voltage and configure the time base by writing to the bit fields in the Control C (ADCn.CTRLC) register:

```
ADC0.CTRLC = ADC_REFSEL_VDD_gc | TIMEBASE_VALUE << ADC_TIMEBASE_gp;
```

Info: Time base is used for timing internal delays in the ADC before starting a conversion, such as the guard time between changing input reference or PGA gain settings. Use the definition in *adc.h* to get a period equal to or greater than 1 µs:

```
#define TIMEBASE_VALUE (uint8_t)ceil(F_CPU*0.000001)
```

`ceil` is used to round up to an integer to write it to the Control C (ADCn.CTRLC) register.

Info: A single-ended ADC will measure the voltage difference between one pin and ground. The voltage reference defines the conversion range of the ADC. Ground is used as zero, while the voltage reference is the highest value within the range. The ADC voltage reference can be selected among different levels between ground and the microcontroller supply voltage to utilize as much of the ADC resolution as possible for inputs in a given voltage range.

Tip: For detailed info on Differential and Single-Ended ADC, refer to the white paper Differential and Single-Ended ADC available from www.microchip.com/DS00003197.

5. Configure the Sample Duration bit field in the Control E (ADCn.CTRLE) register:

```
ADC0.CTRLE = 3;
```

Info: Use Sample Duration to configure the sampling period of the ADC input. The sample duration is given in ADC clock cycles.

6. Enable Free-Running mode in the Control F (ADCn.CTRLF) register:

```
ADC0.CTRLF = ADC_FREERUN_bm;
```

Info: Free-running starts a new conversion when the previous is finished. Free-running requires the first conversion to be initiated before taking effect.

7. Configure the ADC to use the AIN5 input channel by writing the MUXPOS (ADCn.MUXPOS) register:

```
ADC0.MUXPOS = ADC_MUXPOS_AIN5_gc;
```

Info: The device data sheet, section *I/O Multiplexing and Considerations*, indicates which device pins the various analog input channels are connected to. AIN5 is available at pin PA5.

8. Configure the mode of operation for the ADC by writing to the MODE bit field in the COMMAND (ADCn.COMMAND) register:

```
ADC0.COMMAND = ADC_MODE_SINGLE_12BIT_gc;
```

Info: When the ADC is in Single mode, there is only one conversion per trigger.

9. Start conversion by writing to the START bit field in the COMMAND (ADCn.COMMAND) register. Place this in *main.c* before the while loop:

```
ADC0.COMMAND |= ADC_START_IMMEDIATE_gc;
```

10. Verify that the solution builds with no errors by selecting *Build → Build Solution* from the top menu bar in Microchip Studio or pressing the *F7* key.

Result: Code for initializing the ADC is complete, and the expected result is listed below:

```
void adc_init()
{
    ADC0.CTRLA = ADC_ENABLE_bm;
    ADC0.CTRLB = ADC_PRESC_DIV4_gc;
    ADC0.CTRLC = ADC_REFSEL_VDD_gc | TIMEBASE_VALUE << ADC_TIMEBASE_gp;
    ADC0.CTRLE = 3;
    ADC0.CTRLF = ADC_FREERUN_bm;
    ADC0.MUXPOS = ADC_MUXPOS_AIN5_gc;
    ADC0.COMMAND = ADC_MODE_SINGLE_12BIT_gc;
}
```

2.3.1 Transmit ADC Samples

To do: Add code to transmit ADC samples over USART when new results are ready.

1. Open *main.c* and locate the empty while(1) loop.
2. Check if the ADC has a new conversion result ready, and transmit new samples in a data streamer packet. Do this by adding the following code to the empty while(1) loop:

```
if(adc_result_is_ready())
{
    adc_t.adc_sample = adc_get_sample();
    adc_t.adc_result = adc_get_result();

    adc_t.adc_average_result = adc_t.adc_result;

    transmit_to_DV();
}
```

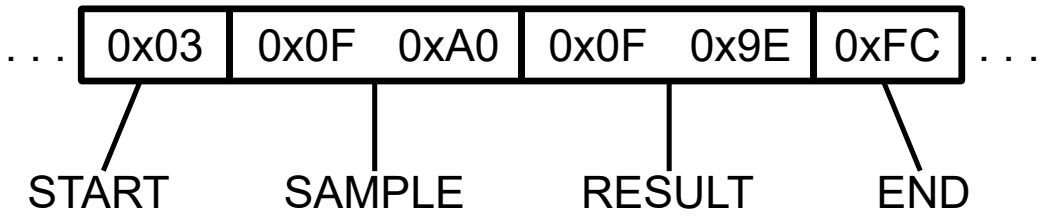
Info: `adc_result_is_ready()`, `adc_get_sample()` and `adc_get_result()` are defined in *adc.c*. `transmit_to_DV()` and the `adc_t` data type are defined in `data_streamer.c` and `data_streamer.h`, respectively:

- `adc_result_is_ready()` returns the ADC's result ready flag. The result ready flag is bit 0 in ADC0.INTFLAGS. This bit is set when a new result is ready from the ADC. The bit is automatically cleared when ADC0.RESULT is read.
- `adc_get_sample()` returns a single ADC conversion stored in the SAMPLE register
- `adc_get_result()` returns the ADC result stored in the RESULT register
- `transmit_to_DV()` is implemented in *data_streamer.c*. This function will transmit a data streamer packet over USART.
- `adc_t` is a struct variable. The structure stores the conversion values to be sent over USART.

3. Verify that the solution builds with no errors by selecting *Build → Build Solution* from the Microchip Studio top menu bar or pressing the *F7* key.
4. Study the implementation of `transmit_to_DV()`.

Info: The packet structure follows the *Data Streamer* protocol. This protocol allows *Data Visualizer* to receive and interpret streamed multi-byte data as well as data originating from multiple sources. The packet is initialized in `transmit_to_DV()`. The packet structure or stream format is defined in Figure 2-3:

Figure 2-3. Assignment 1: Stream Format

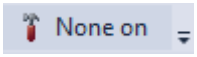


The data stream Stream Format is processed in the same order as the variable group specifies. All data must be given as little-endian values, meaning that the lowest byte must be sent first. Additionally, a wrapper consisting of one byte before and one byte after the data stream variables must be added. This wrapper is used by the interpreter to synchronize to the data stream. The Start byte can be of an arbitrary value, but the end byte must be the inverse of the Start byte.

Result: Code to transmit ADC samples for Single mode has been added to *main.c*.

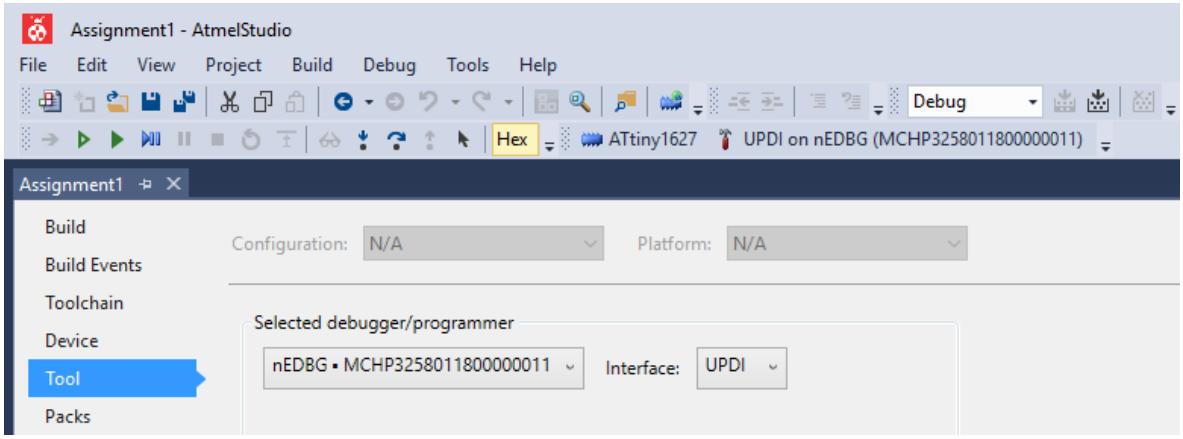
2.3.2 Programming

To do: Program the ATtiny1627 with the *Assignment1* project.

1. Open the *Tool* menu pane by clicking the tool button available in the top menu bar: 

2. Open the *Selected debugger/programmer* drop-down menu and select the tool named *nEDBG MCHPxxx*. The exact tool name is unique for a given tool. Figure 2-4 illustrates an example.

Figure 2-4. Assignment 1: Tool Pane



3. Program the device by selecting *Debug → Start Without Debugging* from the top menu bar.

Info: Microchip Studio will display a prompt to upgrade the embedded debugger firmware if new firmware is available. These firmware upgrades are simple but mandatory.

Result: The ATtiny1627 is programmed.

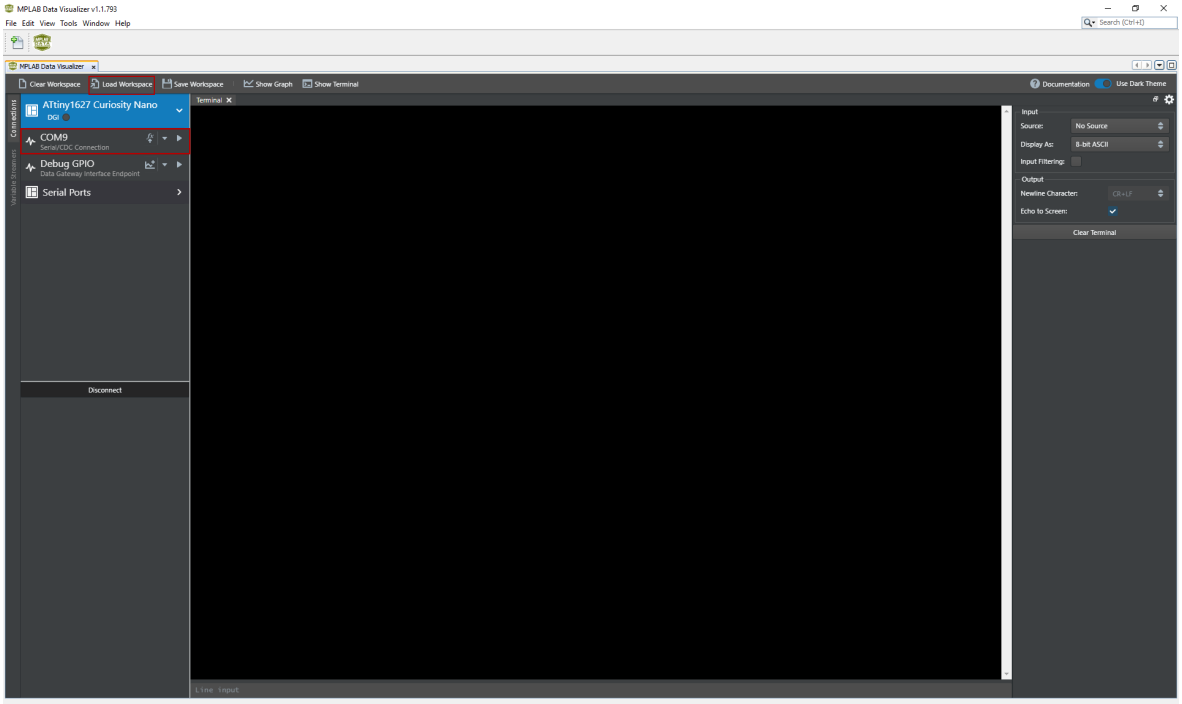
2.3.3 Plot Graph In MPLAB® Data Visualizer

MPLAB *Data Visualizer* is a program used to process and visualize data from a running embedded target. The program may be accessed as an MPLAB X IDE plugin or a stand-alone program. In this assignment, the *Data Visualizer* will be configured to graph ADC acquisitions received over USART. The configuration is done through a saved workspace, and the basics of how to display the data are explained. Click the *Documentation* button in MPLAB *Data Visualizer* for a detailed guide on how to set up your workspace.

To do: Configure MPLAB *Data Visualizer* to graph received ADC samples.

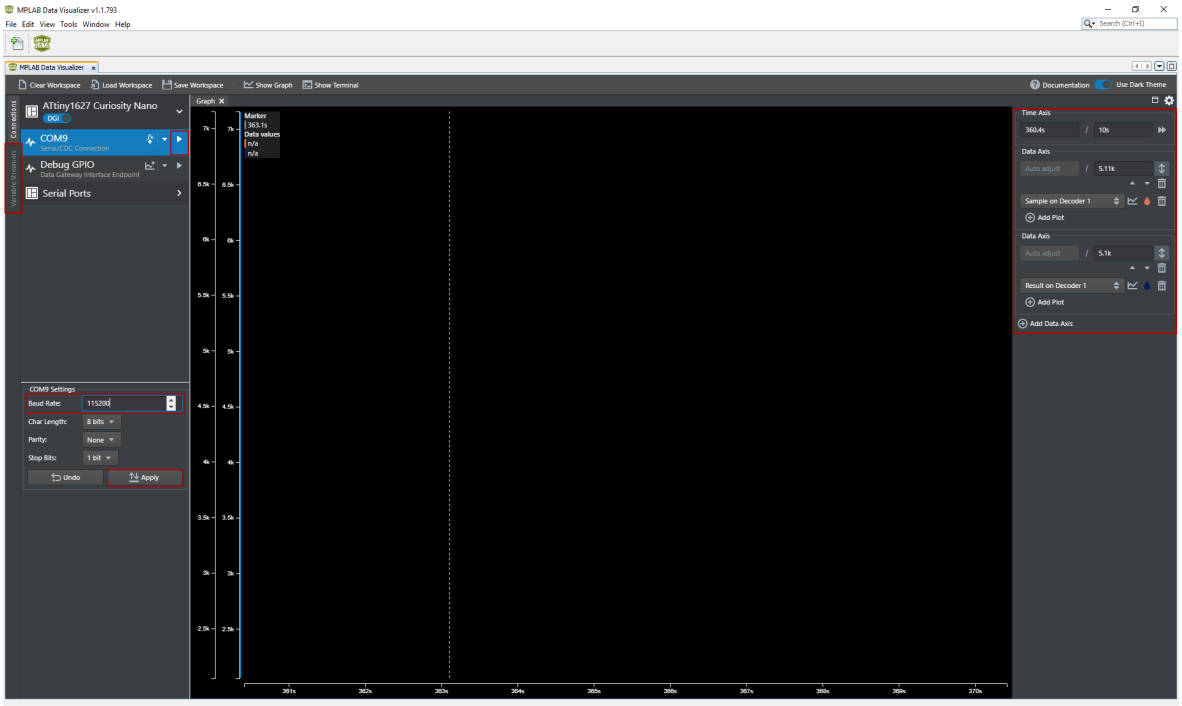
1. Open the program and plug in an already flashed device. Make sure the COM-port used for USART communication is not already in use. The start screen should be similar to Figure 2-5.

Figure 2-5. Assignment 1: MPLAB® Data Visualizer Starting Page



2. Load the workspace. Press the *Load Workspace* button and add the workspace-file called *Assignment1.json*. All the workspaces for this training can be found in `DataStreamerConfiguration`. Two axes should appear in the graph. These can be configured on the panel on the right-hand side, as illustrated in Figure 2-6.

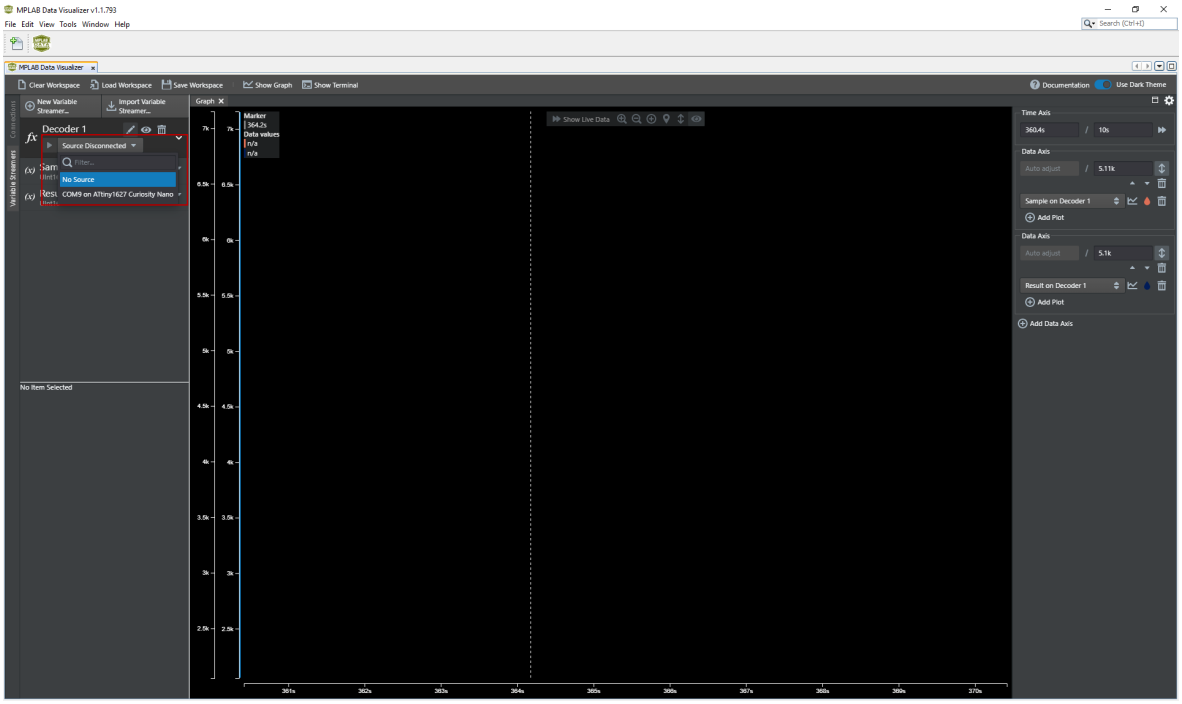
Figure 2-6. Assignment 1: Data Visualizer Setup



3. Plot the data. Choose the COM-port on the left-hand side panel seen in Figure 2-6. Make sure the *Baud Rate* is *115200* and press Apply to set the baud rate. Press the play button on the COM-port to start the USART

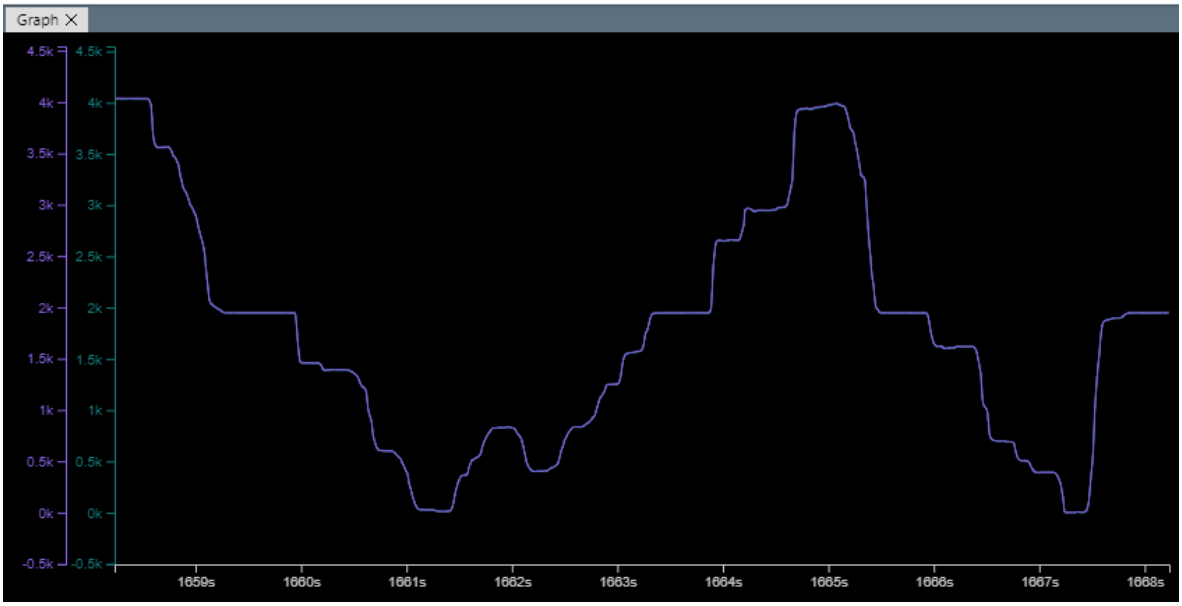
communication. Press the *Variable Streamers* button to connect the decider to the USART data stream. Set the COM port as input to Decoder 1, as shown in Figure 2-7.

Figure 2-7. Assignment 1: Variable Streamers



4. Press *Show Live Data* to start plotting live data from the device. The axes are on auto-range and will scale according to the input from the sensor. Figure 2-8 shows an example of the plotted result when pushing the sensor with different pressure. Since the ADC is 12-bit, the maximum range for the data will be within [0,4095].

Figure 2-8. Assignment 1: Single Mode Plot



Info: The decoder file decodes the data stream from the COM-port. The decoded sources are called Sample and Result. *Sample* is the value placed in the Sample register from a single conversion operation. *Result* is the value placed in the Result register. Depending on the operation, it can either be a single sample or a sum of multiple samples. In Single mode, there is a single conversion per trigger. Therefore, as seen in Figure 2-8, it is expected that the two sources are identical.

Result: The MPLAB *Data Visualizer* has been configured to graph ADC samples received from the nEDBG USART serial gateway.

2.4 ADC Burst Mode

With the setup from Single mode, configuring the ADC to Burst mode requires only a few lines of code to be changed.

To do: Edit `adc_init()` to configure the ADC as listed below:

- 1024 sample accumulation with free running
- ADC sampling in bursts

1. Configure the SAMPNUM bit field in the Control F (ADC0.CTRLF) register:

```
ADC0.CTRLF = ADC_SAMPNUM_ACC1024_gc | ADC_FREERUN_bm;
```

Info: When the ADC is in Burst mode, a configurable number of samples are accumulated into a single ADC result (Sample Accumulation). The RESULT register holds the sum of these accumulations and can be used as an average result of the sample.

2. Configure the mode of operation for the ADC to Burst by writing to the MODE bit field in the COMMAND (ADC0.COMMAND) register:

```
ADC0.COMMAND = ADC_MODE_BURST_gc;
```

3. Verify that the solution builds with no errors by selecting *Build → Build Solution* from the top menu bar in Microchip Studio or pressing the *F7* key.

Result: Code for initializing the ADC is complete, and the expected result is listed below:

```
void adc_init()
{
    ADC0.CTRLA = ADC_ENABLE_bm;
    ADC0.CTRLB = ADC_PRESC_DIV4_gc;
    ADC0.CTRLC = ADC_REFSEL_VDD_gc | TIMEBASE_VALUE << ADC_TIMEBASE_gp;
    ADC0.CTRLE = 3;
    ADC0.CTRLF = ADC_SAMPNUM_ACC1024_gc | ADC_FREERUN_bm ;
    ADC0.MUXPOS = ADC_MUXPOS_AIN5_gc;
    ADC0.COMMAND = ADC_MODE_BURST_gc;
}
```

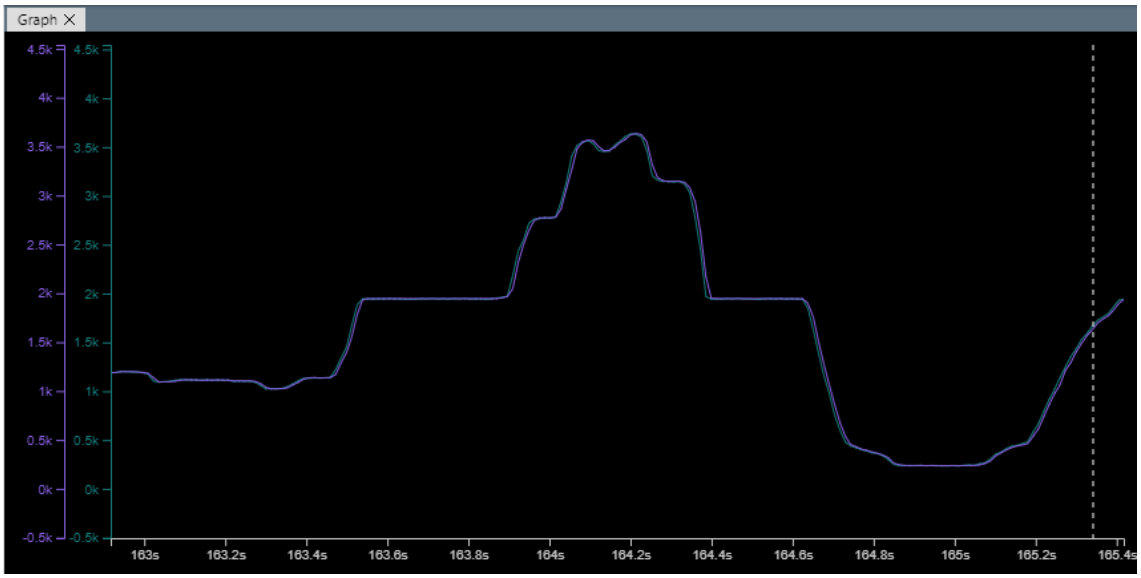
2.4.1 Burst Mode Results

To do: Plot the result for Burst mode using the *Data Visualizer*.

1. Replace the corresponding line of code in *main.c* to get the average result from the accumulated samples:
```
adc_t.adc_average_result = adc_t.adc_result>>ADC_SAMPLES;
```
2. Flash the device and plot the results in the *Data Visualizer*. Figure 2-9 shows the results from Burst mode.

Result: Following the same procedure as explained for Single mode, the result for Burst mode is plotted in the *Data Visualizer*:

Figure 2-9. Assignment 1: Burst Mode Plot



The sample-signal in green is the same as for Single mode. It plots the single conversion held in the sample register. The result-signal in purple will no longer be the same and will now be an average of the accumulated samples. The plots are still almost identical except for instances with very sudden changes in input. The difference is more noticeable when looking at a short time intervals, as shown in Figure 2-9.

3. Assignment 2: ADC Accumulation Modes

In this assignment the ADC of the ATtiny1627 will be configured to sample the Force Click board using the Burst and Series Conversion mode and conversions will be triggered by the RTC overflow event. The ATtiny1627 will go to sleep between samples to reduce power consumption.

The sampled ADC data will be transmitted over USART to the *Data Visualizer*. The *Data Visualizer* will be configured to graph single samples and the accumulated samples to visualize how the different configurations affect the noise and aliasing of the signal.

The Microchip Studio Solution *Assignment2* includes the required starting point for this assignment and should be available in the training material folder at the following path: `Assignment2`.

- **Objectives**
  - Configure and run the ADC using Burst Conversion mode and the RTC overflow event as conversion trigger
  - Configure and run the ADC using Series Conversion mode and the RTC overflow event as conversion trigger
  - Configure the ATtiny1627 to go to sleep between ADC measurements to save power

3.1 ADC Burst Mode With Event Trigger

To do:

- Edit the `adc_init()` so that the ADC is in Burst mode and starts conversion on the event trigger
- Study the `rtc_init()` and `event_system_init()` so it is clear how they are configured

Info:
The Event System (EVSYS) enables direct peripheral-to-peripheral signaling. It allows a change in one peripheral (the event generator) to trigger actions in other peripherals (the event users) through event channels, without using the CPU. The RTC can be used to generate an event every time the timer overflows, and the ADC can be configured as a user that starts a conversion every time it receives an event.

1. Change the `adc_init()` so that:

```
ADC0.COMMAND = ADC_MODE_BURST_gc | ADC_START_EVENT_TRIGGER_gc;
```

2. Study the `rtc_init()` and `event_system_init()` in *rtc.c* and *event_system.c*, respectively. Make sure you understand how they are configured.

Info: The RTC is configured to run using the 32.768 kHz ULP internal oscillator and a prescaler of 1. The RTC period is calculated using the RTC clock frequency and the desired ADC result frequency `ADC_RESULT_FREQ` of 64 Hz.

Info: The RTC overflow event is used as an event generator for event channel 0, and the ADC is listening to channel 0.

3. Verify that the solution builds with no errors by selecting *Build* → *Build Solution* from the top menu bar in Microchip Studio or pressing the *F7* key.

4. Flash the device by selecting *Debug → Start without debugging* for the top menu bar in Microchip Studio or pressing the *Ctrl+Alt+F5* keys.
5. Plot the single ADC sample vs. the ADC average result using the MPLAB *Data Visualizer* by loading workspace *Assignment2.json*. Try changing the number of ADC samples by changing the `ADC_SAMPLES` macro in *adc.h*. to see what effect it has on the noise level of the signal.

Tip: You can visualize the ADC result frequency by looking at how fast LED0 is blinking.

Tip: To get a better view of the noise of the signals it is recommended to set the range and offset for both axis to 16 and 10 and apply no force on the sensor.

Result: Measuring the analog signal from the Force click using the ADC in Burst mode and using the RTC and event system to continuously trigger the ADC accumulation. The results should look similar to Figure 3-1 when the force sensor is pressed with varying force. When investigating the noise, it should look similar to Figure 3-2, where the force sensor is not pressed, and the range and offset is set to 16 and 10, respectively.

Figure 3-1. Assignment 2: Varying Force Applied To Force Sensor

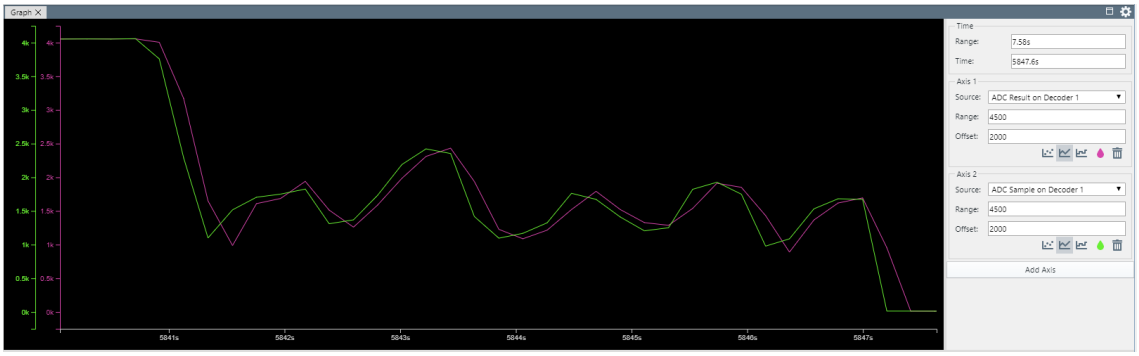
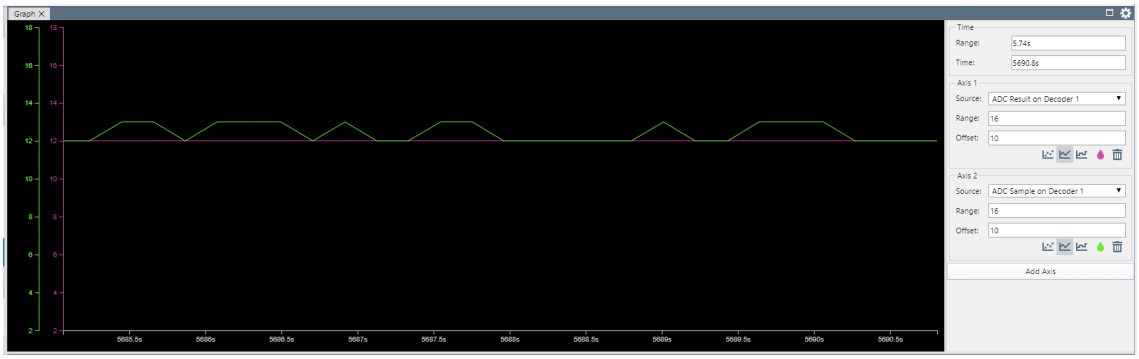


Figure 3-2. Assignment 2: Noise Investigation



3.2 ADC Series Mode With Event Trigger

To do:

- Edit the `adc_init()`, so the ADC uses Series Conversion mode and works while the device is in Standby sleep mode
- Change the RTC period by editing `rtc_init()`
- Edit the program so the ATtiny1627 can go to Standby sleep mode

1. Configure the ADC for Series Conversion mode to be used in Standby sleep mode.
   1.1. Change the Conversion mode by changing the Command register:

```
ADC0.COMMAND = ADC_MODE_SERIES_gc | ADC_START_EVENT_TRIGGER_gc;
```

   1.2. Make the ADC trigger an ADC result ready interrupt to wake up the CPU from sleep every time an ADC result is ready by adding the following line to the `adc_init()`:

```
/*Activate interrupt on Result ready to wake up from sleep*/
ADC0.INTCTRL = ADC_RESRDY_bm;
```

   1.3. Change the following code to `adc_init()` to enable the ADC to run in Standby sleep mode:

```
ADC0.CTRLA = ADC_ENABLE_bm | ADC_RUNSTDBY_bm; /*Enable the ADC and run in
standby sleep mode*/
```

   1.4. Change the `adc_result_is_ready()` to the following, so it works with an interrupt based ADC:

```
bool adc_result_is_ready()
{
    if(adc_result_is_ready_flag){
        adc_result_is_ready_flag = 0;
        return true;
    }
    return false;
}
```

2. Change the `rtc_init()` in *rtc.c*, so the RTC period is dependent on the ADC sampling frequency and not the ADC result frequency by writing:

```
RTC.PER = (RTC_CLOCK/(float)(ADC_SAMPLING_FREQ))+0.5;
```

Info: When the ADC is in Series mode, each event triggers one single sample as opposed to the Burst mode, where one event triggers all the samples needed for one result, which means that the sampling frequency must be a lot higher than the result frequency.

Tip: To control the result frequency instead of the sampling frequency, define the sampling frequency using the result frequency and the number of samples by writing:

```
#define ADC_SAMPL_FREQ (uint16_t)(ADC_RESULT_FREQ << ADC_SAMPLES)
```

3. Configure ATtiny1627 so it can go to standby sleep and still retain its previous functionality.
   3.1. Study the `slpctrl_init()` in *sleep_control.c* to understand how the sleep controller is configured.
   3.2. Add `slpctrl_init()` to the initialization sequence in the main function.
   3.3. Add `sei()` to the end of the initialization sequence in the main function to enable interrupts so the device can wake up from Standby sleep mode.

3.4. Add `sleep_cpu()` to the while function so the device will go to sleep between each result:

```
while (1)
{
    sleep_cpu();
    if(adc_result_is_ready())
    {
        adc_t.adc_sample = adc_get_sample();
        adc_t.adc_result = adc_get_result();
        adc_t.adc_average_result = adc_t.adc_result>>ADC_SAMPLES;
        trasmit_to_DV();
        PORTB.OUTTGL = LED0_bm;
    }
}
```

3.5. Add the following code to the end of the `transmit_to_DV()` in *data_streamer.c* to make sure the device does not go to sleep before the transmission is complete:

```
while (!(USART0.STATUS & USART_TXCIF_bm)); /* Wait for all USART transmission to
be finished */
USART0.STATUS |= USART_TXCIF_bm; /* Clear Tx complete flag */
```

Info: When the device does not go to sleep, it uses 24 mW, while when it goes to Standby sleep mode, it uses 12 mW, so it can be seen that going to sleep between results can reduce the power by up to 50%. The measurements were taken with a result frequency of 20 Hz, 512 accumulated samples, without the blinking LED.

4. Verify that the solution builds with no errors by selecting *Build → Build Solution* from the top menu bar in Microchip Studio or pressing the *F7* key.
5. Flash the device by selecting *Debug → Start without debugging* for the top menu bar in Microchip Studio or pressing the *Ctrl+Alt+F5* keys.
6. Plot the single ADC sample vs. the ADC average result using the MPLAB *Data Visualizer*. Try different sampling frequencies and change the force applied to the force sensor periodically.

Info: Due to the configured clock frequency of the RTC, the highest sampling frequency is 32768 Hz. The result frequency is dependent on the sampling frequency and the number of samples, meaning that the maximum result frequency for 1024 samples is 32 Hz.

Result: Measuring the analog signal from the Force click using the ADC in Series mode and using the RTC and event system to continuously trigger the ADC sample. The device is in Standby sleep mode between each result. When the sampling frequency is low, the result should look similar to Figure 3-3, and when it is high, it should look similar to Figure 3-4. See that, if the sampling frequency is too low, the result becomes the average of the rapidly changing signal, and a lot of information is lost.

Figure 3-3. Assignment 2: 1 Hz Result Frequency

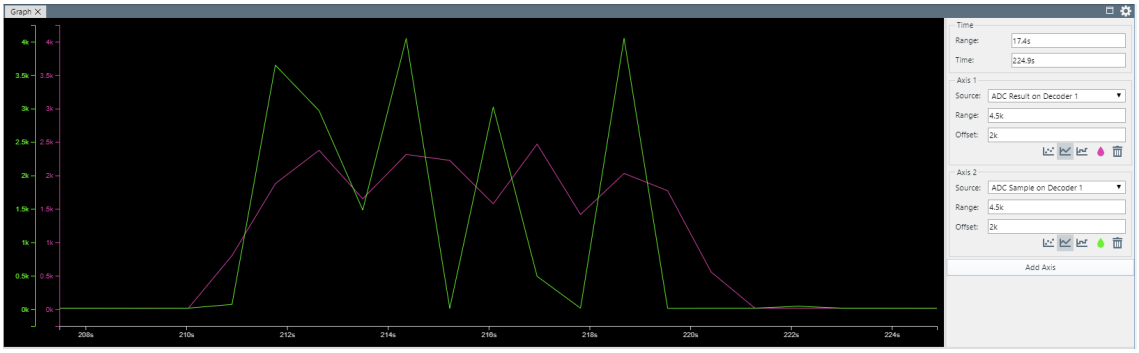
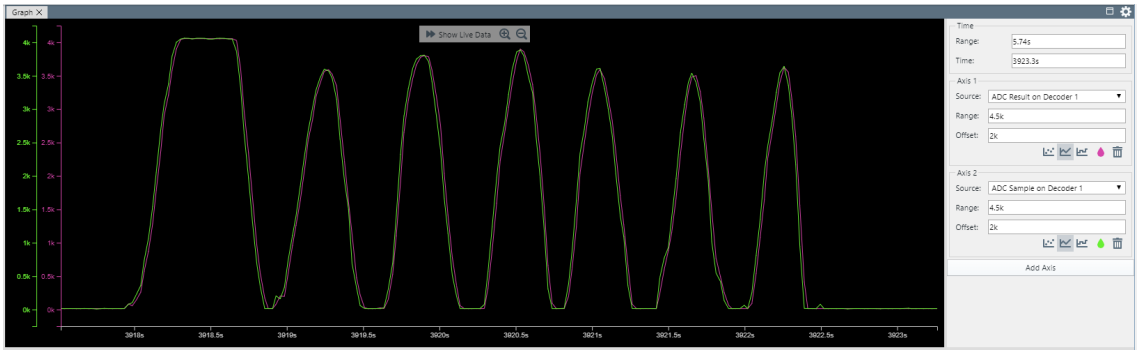


Figure 3-4. Assignment 2: 64 Hz Result Frequency



4. Assignment 3: 17-Bit Resolution

In this assignment, the ADC of the ATtiny1627 will be configured to sample the DAC reference from the AC module with a 17-bit resolution. The ADC will be in Burst mode, and a sample will be triggered by an RTC overflow event as in assignment 2.

Typical use cases where high-resolution ADC is useful:

- Strain gauges (can be used to measure weight)
- Thermocouples

The sampled ADC data will be transmitted over USART to the *Data Visualizer*. The *Data Visualizer* will be configured to graph the 12-bit resolution against the 17-bit resolution measurement.

The Microchip Studio Solution *Assignment3* includes the required starting point for this assignment and should be available in the training material folder at the following path: `Assignment3`.

- **Objectives**
  - Understand how oversampling can be used to increase the resolution of a measurement
  - Increase the resolution of the ADC to 17-bits by oversampling

4.1 ADC 17-Bit Resolution

To do:

- Understand how to increase the resolution of the ADC measurement by oversampling
- Setup DAC reference voltage used by the AC as input to the ADC
- Configure the ADC to have 17-bit resolution using oversampling

Info: To increase the ADC resolution by n bits, the signal must be oversampled by 4^n samples. To achieve a 17-bit resolution, an additional 5-bit resolution is needed, which means that the signal needs to be sampled $4^5 = 1024$ times. The ADC can be configured to accumulate up to 1024 samples. The accumulated result will be the sum of the 1024 samples. This result needs to be scaled down to achieve the desired bit width by dividing by the scaling factor. The scaling factor is given by 2^n. To increase the resolution by 5 bits, we either need to divide the result by $2^5 = 32$ or right shift it by 5.

1. At the beginning of `adc_init`, set up the DAC voltage reference to 2.048V by writing:

```
VREF.CTRLA = VREF_AC0REFSEL_2V048_gc; /* Voltage reference for AC */
AC0.DACREF = 0xFF; /* DACREF is VREF */
```

Tip: The internal reference voltage generator (DACREF) can be used to create specific voltage references. The voltage reference V_{DACREF} is configured through the Analog Comparator.

Info:
The DAC voltage reference depends on the DACREF value and the reference voltage selected in the V_{REF} module Control Register A and is calculated as: $V_{DACREF} = \frac{DACREF}{256} \times V_{REF}$

2. Configure the MUXPOS to have the DAC reference as an input source:

```
ADC0.MUXPOS = ADC_MUXPOS_DAC_gc; /*DAC from AC0*/
```

3. Configure the ADC so it can achieve 17-bit resolution by setting `ADC_SAMPLES` in `adc.h` to the right number of samples using the `ADC_SAMPNUM` macros by writing:

```
#define ADC_SAMPLES ADC_SAMPNUM_ACC1024_gc
```

4. In the `main()` function right shift the accumulated result to scale the result down to 17-bit by writing:

```
adc_t.adc_high_res_result = adc_t.adc_result>>(ADC_SAMPLES/2);
```

5. Left-shift the averaged result, so it becomes on the same scale as the high-resolution result by writing:

```
adc_t.adc_average_result = adc_t.adc_average_result<<(ADC_SAMPLES/2);
```

Info: To see the impact of the difference in resolution between 12-bit and 17-bit, the two results are scaled to the same range.

Result: The main should look like the following:

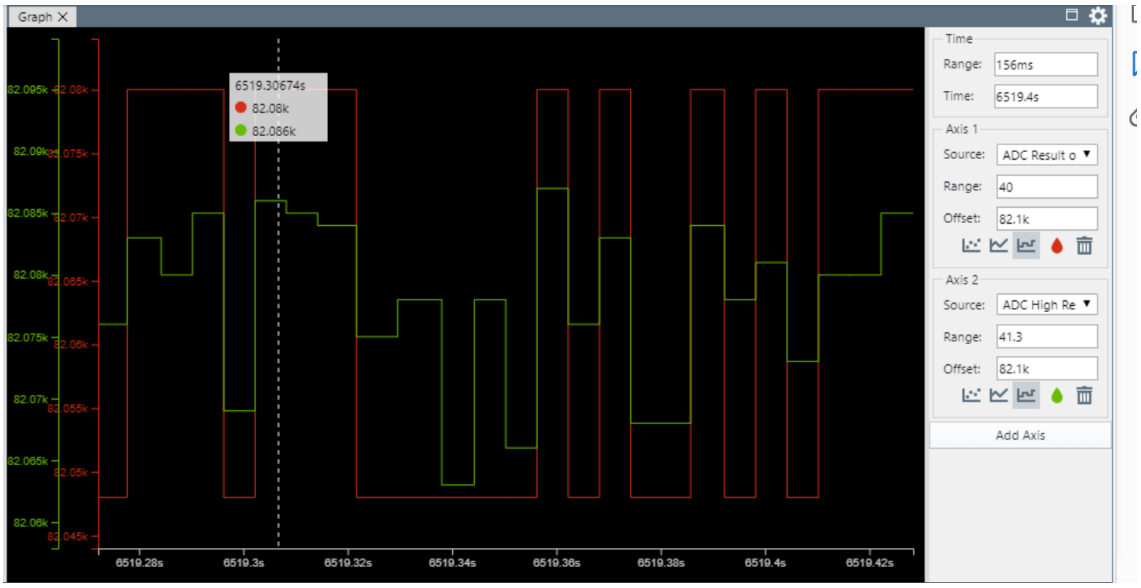
```
adc_t.adc_average_result = adc_t.adc_result>>ADC_SAMPLES;
adc_t.adc_average_result = adc_t.adc_average_result<<(ADC_SAMPLES/2);

adc_t.adc_high_res_result = adc_t.adc_result>>(ADC_SAMPLES/2);
```

It is important to take the average of the accumulated measurements before the 12-bit result is scaled up.

6. Verify that the solution builds with no errors by selecting *Build → Build Solution* from the top menu bar in Microchip Studio or pressing the *F7* key.
7. Flash the device by selecting *Debug → Start without debugging* for the top menu bar in Microchip Studio or pressing the *Ctrl+Alt+F5* keys.
8. Plot the 17-bit result and the 12-bit result in the *Data visualizer* by loading the workspace *Assignment3.json.*

Result: The ADC is configured to measure the DAC reference voltage from the AC module using oversampling to increase the resolution to 17-bit. Figure 4-1 shows the comparison between the 17-bit and the 12-bit result. The 17-bit result is in green, while the 12-bit result is in red. Observe that the 17-bit signal can take much smaller steps than the 12-bit signal.

Figure 4-1. Assignment 3: 17-Bit vs. 12-Bit



5. Assignment 4: Read Temperature

In this assignment, temperature measurement with the ADC will be demonstrated. The ATtiny1627 has an on-chip temperature sensor available. This sensor allows for basic temperature sensing, and no extra hardware is needed.

A temperature measurement can be done by initializing the ADC, selecting internal reference, selecting the temperature sensor as input, and acquiring the data by running a 12-bit single-ended conversion.

The Microchip Studio Solution *Assignment4* includes the required starting point for this assignment and should be available in the training material folder at the following path: `Assignment4`.

- **Objectives**
  - Configure the ADC for temperature measurement with the on-chip temperature sensor
  - Find the results through debugging in *Microchip Studio*

5.1 Setup Tempsense

To do:

- Edit `adc_init()` to configure ADC in Single mode for temperature measurement
- Find the results by debugging the code and adding breakpoints at the temperature variables

1. Configure the reference voltage to the one recommended in the data sheet for the internal temperature sensor:

```
ADC0.CTRLC = ADC_REFSEL_1024MV_gc | (TIMEBASE_VALUE << ADC_TIMEBASE_gp);
```

Info: The measured voltage has an almost linear relationship with the temperature. Due to process variations, the temperature sensor output voltage varies between individual devices at the same temperature. The compensation factors determined during the production test are stored in the Signature Row. These compensations factors are generated for the internal 1.024V reference.

2. Configure the ADC to use the internal temperature sensor by writing to the MUXPOS register:

```
ADC0.MUXPOS = ADC_MUXPOS_TEMPSENSE_gc;
```

3. Configure sample duration using the variable *TEMPSENSE_SAMPDUR*:

```
ADC0.CTRLE = TEMPSENSE_SAMPDUR;
```

Info: The variable calculates the sample duration for the temperature sensor. As per the data sheet, Sample Duration should be greater than or equal to $32\ \mu s \times f_{CLK\_ADC}$, expressed in the code below:

```
#define TEMPSENSE_SAMPDUR ((uint8_t) ceil(F_CPU*0.000032/2))
```

The division by two is because $f_{CLK\_ADC} = f_{CLK\_CPU}/2$.

4. Configure the ADC prescaler to produce the desired CLK_PER/2 ADC clock by writing the CTRLB register:

```
ADC0.CTRLB = ADC_PRESC_DIV2_gc;
```

Result: Code for initializing the ADC is complete, and the expected result is listed below:

```
void adc_init()
{
    ADC0.CTRLA = ADC_ENABLE_bm;
    ADC0.CTRLB = ADC_PRESC_DIV2_gc;
    ADC0.CTRLC = ADC_REFSEL_1024MV_gc | (TIMEBASE_VALUE << ADC_TIMEBASE_gp);

    ADC0.CTRLE = TEMPSENSE_SAMPDUR;
    ADC0.MUXPOS = ADC_MUXPOS_TEMPSENSE_gc; /* ADC Internal Temperature Sensor */
    ADC0.COMMAND = ADC_MODE_SINGLE_12BIT_gc; /* Single 12-bit mode */
}
```

5. Add calibration variables for the measurement. Place the variables before the `while(1)` loop:

```
int8_t sigrow_offset = SIGROW.TEMPSENSE1;
uint8_t sigrow_gain = SIGROW.TEMPSENSE0;
```

Info:
The content of the Signature Row fuses (SIGROW) is preprogrammed and cannot be altered. SIGROW holds information such as device ID, serial number, and calibration values.

The Temperature Sensor Calibration registers contain correction factors for temperature measurements from the on-chip sensor. SIGROW.TEMPSENSE0 is a correction factor for the gain/slope (unsigned) and SIGROW.TEMPSENSE1 is a correction factor for the offset (signed).

6. Add the code below to the main while loop to measure temperature in Kelvin and Celcius:

```
        adc_reading = ADC0.RESULT >> 2; /* 10-bit MSb of ADC result with 1.024V internal
reference */
        uint32_t temp = adc_reading - sigrow_offset;
        temp *= sigrow_gain; /* Result might overflow 16-bit variable (10-bit + 8-bit) */
        temp += 0x80; /* Add 256/2 to get correct integer rounding on division below */
        temp >>= 8; /* Divide result by 256 to get processed temperature in Kelvin */
        temperature_in_K = temp;
        temperature_in_degC = temperature_in_K - 273;
```

Info: The temperature measurement uses the following formula:

$$T = \frac{(\text{ADC Result} - \text{Offset Correction}) \times \text{Gain Correction}}{256}$$

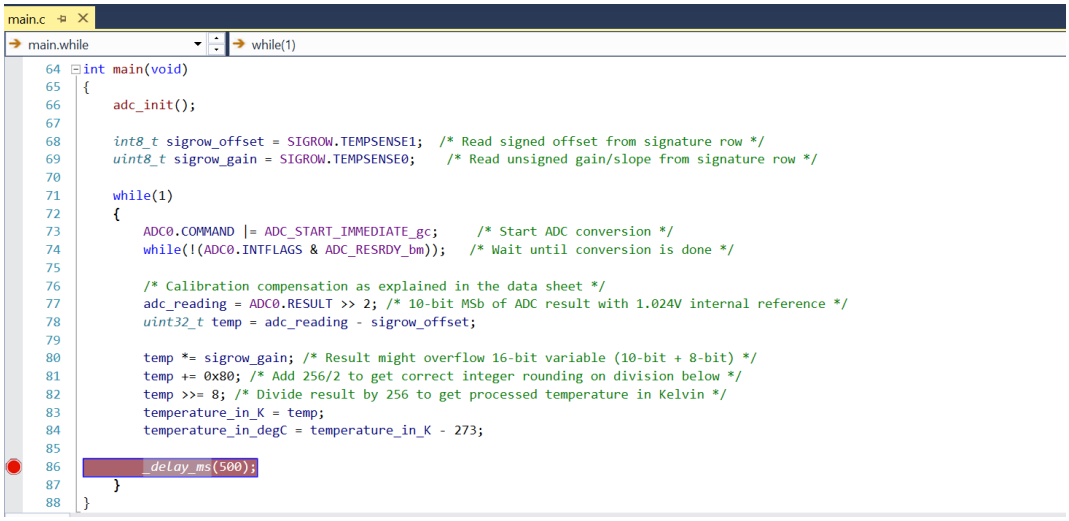
7. Verify that the solution builds with no errors by selecting *Build → Build Solution* from the top menu bar in Microchip Studio or pressing the F7 key.
8. Set a breakpoint on the `_delay_ms` to be able to watch `temperature_in_K` and `temperature_in_K`. Figure 5-1 shows the breakpoint as red dots on the left pane of the window.

Tip: Breakpoints can be added by clicking on the left pane to the left of where the line of code is.

9. Flash the device by selecting *Debug → Continue* for the top menu bar in Microchip Studio or pressing the F5 key.
10. Select and right-click `temperature_in_K` and `temperature_in_degC`, and choose *Add watch.*
11. Go through the code by pressing the F5 key and watch as the values of the variables change.

Result: Code for initializing the ADC is complete and the expected result is shown in Figure 5-1:

Figure 5-1. Assignment 4: Temperature Sensor Debugging



Tip: To toggle the displayed value between decimal and hexadecimal, right-click a variable and press *Hexadecimal Display*.

6. Assignment 5: Differential ADC

In this assignment, the Differential mode of the ADC will be demonstrated. In Single-Ended mode, the ADC reads the voltage of a single selectable input source, while in Differential mode, the ADC reads the voltage difference between two input sources.

For this assignment, the Force Click board will be used as one input source, while the other input will be the internal voltage reference AC0 DACREF.

The Microchip Studio Solution *Assignment5* includes the required starting point for this assignment and should be available in the training material folder at the following path: `Assignment5`.

- **Objectives**
  - Configure the ADC for Differential mode using the Force Click board as MUXPOS and DACREF0 as MUXNEG
  - Use the *Data Visualizer* to graph received ADC results

6.1 ADC Differential Mode Setup

To do:

- Edit `adc_init()` to configure ADC in Differential mode, with the Force Click board as MUXPOS and DACREF0 as MUXNEG
- Transmit the data to the *Data Visualizer*

1. Set up the DAC voltage reference to about half of the Force Click board max voltage or VDD/2:

```
VREF.CTRLA = VREF_AC0REFSEL_2V048_gc; /* Voltage reference for AC */
AC0.DACREF = 0xCE; /* Will generate V_DACREF = 1.65V */
```

2. Set up the ADC voltage reference:

```
ADC0.CTRLC = ADC_REFSEL_2048MV_gc | ((uint8_t)TIMEBASE_VALUE << ADC_TIMEBASE0_bp);/*ADC
voltage reference*/
```

Info:
The ADC voltage reference sets the upper limit for the ADC input range. Any values above this range will be interpreted as V_{REF}. In Differential mode, the upper limit is for the difference between two inputs and not the individual inputs. Therefore, $V_{MUXPOS} - V_{MUXNEG} \leq V_{REF}$ will be interpreted correctly, even if $V_{MUXPOS} > V_{REF}$.

3. Configure the MUXPOS register to have the Force Click board as an input source:

```
ADC0.MUXPOS = ADC_MUXPOS_AIN5_gc;
```

4. Configure the MUXNEG to have the DAC reference as an input source:

```
ADC0.MUXNEG = ADC_MUXNEG_DAC_gc;
```

Info:
The input signal from MUXPOS will now be measured in relation to MUXNEG. In Single-Ended mode, MUXPOS is measured in relation to ground.

5. Configure the ADC in Differential mode, with the 12-bit Single Operational mode:

```
ADC0.COMMAND |= ADC_DIFF_bm | ADC_MODE_SINGLE_12BIT_gc;
```

6. Add the following code in `main.c` to calculate the voltage difference between MUXPOS and MUXNEG. The calculation is done in the `while(1)` loop:

```
/* Formula shows how much MUXPOS and MUXNEG differ from each other in Volt */
adc_t.adc_differential_volt = (float)((adc_t.adc_sample * 2.048) / ADC_MAX_VALUE);
```

Info:
For Differential mode, the data format is two's complement with sign extension. The data type of the sample variable should be int16_t when using Differential mode. The data type of the result variable should be int32_t when using Differential mode. Float is used here to calculate signed decimal numbers.

7. Verify that the solution builds with no errors by selecting *Build → Build Solution* from the top menu bar in Microchip Studio or pressing the *F7* key.
8. Flash the device by selecting *Debug → Start without debugging* for the top menu bar in Microchip Studio or pressing the *Ctrl+Alt+F5* keys.
9. Verify hardware connections. Make sure the Force Click board is connected to socket 3 on the Curiosity Nano Adapter.
10. Plot the single ADC sample and voltage difference using the *Data Visualizer* by loading the workspace *Assignment5.json*. Try changing the DAC voltage reference and study its effect on the plotted range.

Tip: To change V_{DACREF}, either change the DACREF number or the voltage reference for AC. This is described in step 1 of Assignment 3.

Info:
Since MUXNEG now is connected to the voltage reference 1.65V, as seen in Figure 6-1 rather than GND, the new zero for the MUXPOS input is 1.65V. For the Force Click board with a range of [0,3.3V], the ADC output will be from [-1.65,1.65V]. See Figure 6-1.

Result: The ADC is running in Differential mode, using the 12-bit Single Operational mode. The difference between MUXPOS and MUXNEG is measured and plotted in the *Data Visualizer*. The results should look similar to Figure 6-1 when the button is pressed with varying force. The figures show measurement using different constant voltages for MUXNEG. Figure 6-1 uses V_{DACREF} = 1.65V. In Figure 6-2, MUXNEG has the input source V_{DACREF} = 1.024V.

Figure 6-1. Assignment 5: Differential Mode With V_{DACREF} = 1.65V

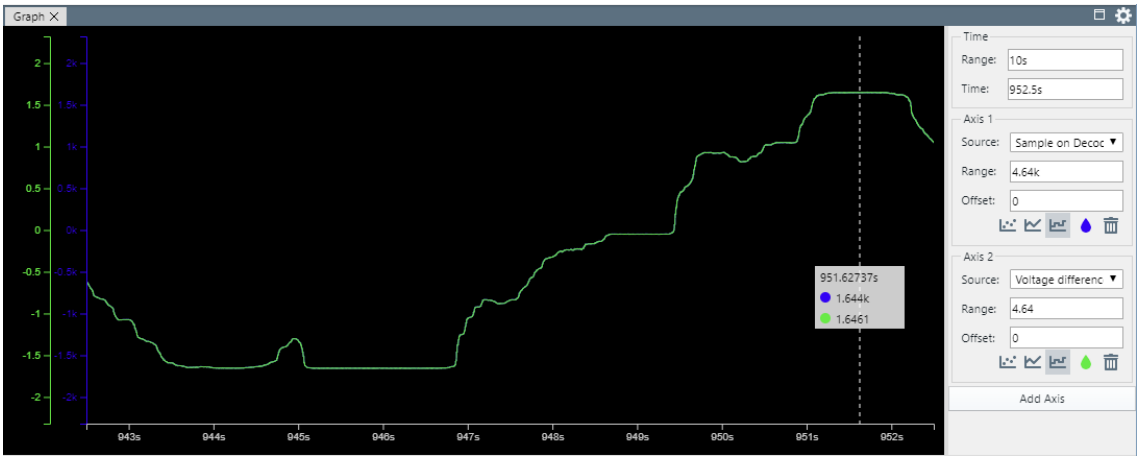
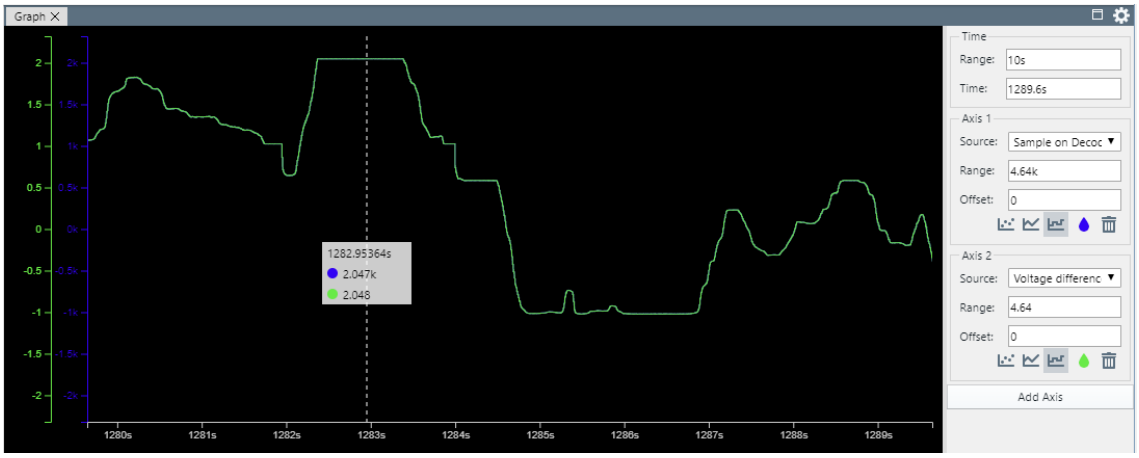


Figure 6-2. Assignment 5: Differential Mode With V_{DACREF} = 1.024V



Info:
By setting the V_{DACREF} to 1.024V, the measurements are shifted upwards in the measured range. The resulting range is from about [-1.024V, 2.048V]. This range is too small for the Force Click board since the highest difference is 2.276V with the range [0,3.3V]. All the values above 2.048V will therefore be interpreted as 2.048V by the ADC. The clipping that occurs is visible in Figure 6-2, where the top value never exceeds 2.048V.

A convenient property of the differential measurement is that since the signal is measured in relation to DACREF instead of ground, it is possible to adjust DACREF to be in the middle of the expected input for MUXPOS. The necessary range can therefore be tailored to a specific signal if the bounds are known. This can allow for higher gain and lowering the voltage reference used by the ADC.

7. Assignment 6: Using Programmable Gain Amplifier (PGA)

In this assignment, the ADC of the ATtiny1627 will be configured to use the programmable gain amplifier (PGA) to amplify the signal from the board. The signal from the Force Click is divided down using two resistors to create a voltage divider. The ADC will be in Burst mode, and a sample will be triggered by an RTC overflow event as in assignment 2.

The sampled ADC data will be transmitted over USART to the *Data Visualizer*. The *Data Visualizer* will first be used to plot the original signal and then plot the amplified signal.

The Microchip Studio Solution *Assignment6* includes the required starting point for this assignment and should be available in the training material folder at the following path: `Assignment6`.

- **Objectives**
  - Setup of the voltage divider
  - Configure the ADC to use the PGA to amplify the signal

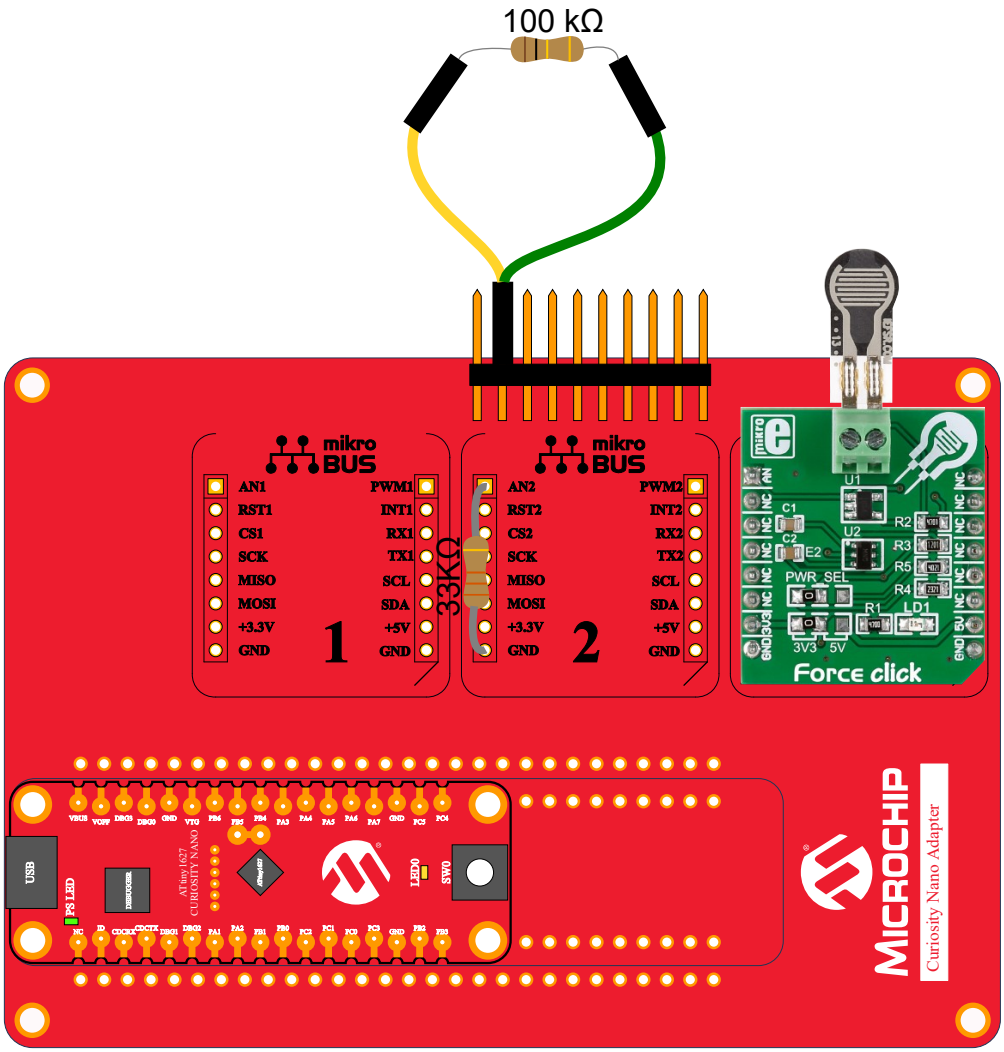
7.1 Hardware Setup

To do:

- Setup of the voltage divider

Info: The voltage divider is created by connecting AN3 and AN2 through the 100 kΩ resistor and connecting AN2 to GND through the 33 kΩ resistor. The new measuring point will then be AN2. If you look at the Curiosity Nano baseboard, you will see that AN2 is connected to PA6, which means that input to the ADC should be AIN6.

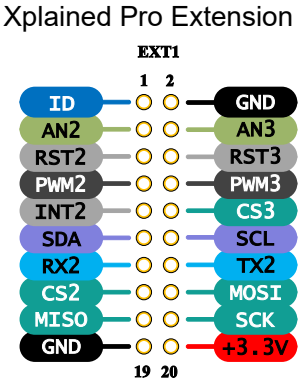
Figure 7-1. Assignment 6: Hardware Setup



1. Connect a 100 kΩ resistor to pin 3 and 4 of the Xplained Pro extension as shown in Figure 7-1. Figure 7-2 shows the extension pinout.
2. Connect the 33 kΩ resistor to AN2 and GND in the mikroBUS 2 header as shown in Figure 7-1.

Info: The voltage divider will divide the voltage by 4, which can be seen as 33 kΩ/(33 kΩ + 100 kΩ) ≈ 0.25.

Figure 7-2. Assignment 6: Xplained Pro Extension Header



7.2 PGA Configuration

To do:

- Plot a graph of measurements without using the PGA
- Configure the ADC to use the PGA to amplify the signal
- Plot a graph of measurements with the amplified signal

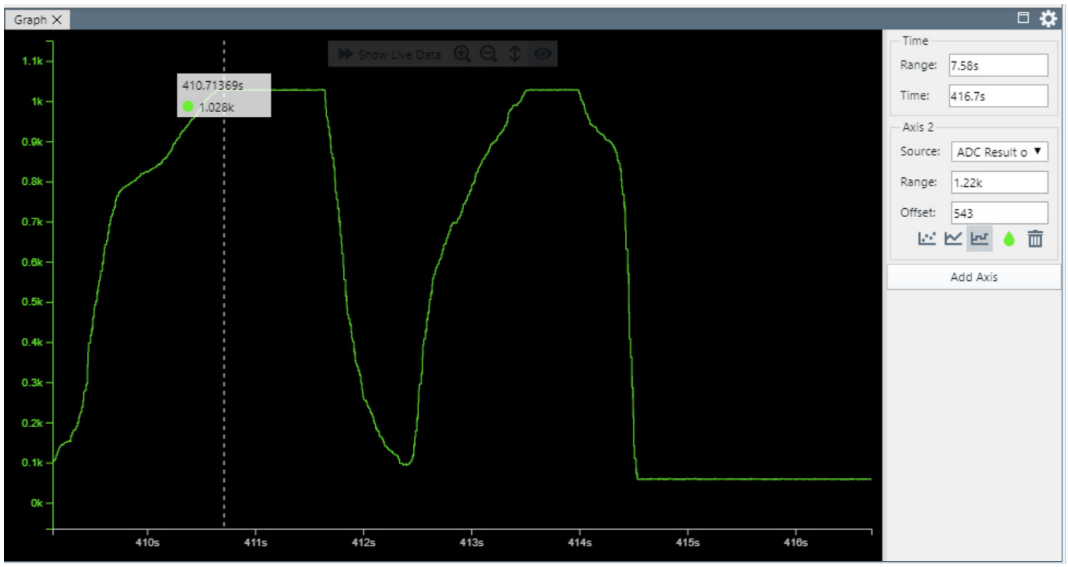
1. Verify that the solution builds with no errors by selecting *Build → Build Solution* from the top menu bar in Microchip Studio or pressing the F7 key.
2. Flash the device by selecting *Debug → Start without debugging* for the top menu bar in Microchip Studio or pressing the *Ctrl+Alt+F5* keys.

Info: Configure the project to do periodic burst measurements using the RTC and event system as in assignment 2.

3. Load the workspace *Assignment6.json* in the MPLAB *Data Visualizer.*
4. Press the force sensor as hard as possible and observe the graph in the MPLAB *Data Visualizer.*

Result: The plot should look similar to Figure 7-3, where the top value is around 1k. The value is expected to be around 1k, as the voltage divider divides the voltage by 4, and V_{CC} is represented in the ADC by 4095.

Figure 7-3. Assignment 6: No Amplification



5. Configure the PGA to amplify the signal by 4x, have a bias current of 50%, a sampling duration of 15 clock cycles, and enable the PGA by writing the following in *adc.c*:

```
ADC0.PGACTRL = ADC_GAIN_4X_gc | ADC_PGABIASSEL_1_2X_gc | ADC_ADCPGASAMPDUR_15CLK_gc |
ADC_PGAEN_bm;
```

Info: The `PGABIASSEL` determines how much current the PGA uses. How much current the PGA needs depends on the ADC clock frequency. See the PGA Control (PGACTRL) register description in the device data sheet for info about how much current the PGA needs.

Info: When using the PGA, the `SAMPDUR` determines the input sample duration, while `ADCPGASAMPDUR` determines for how long the ADC samples the PGA.

6. Connect the input via the PGA by writing:

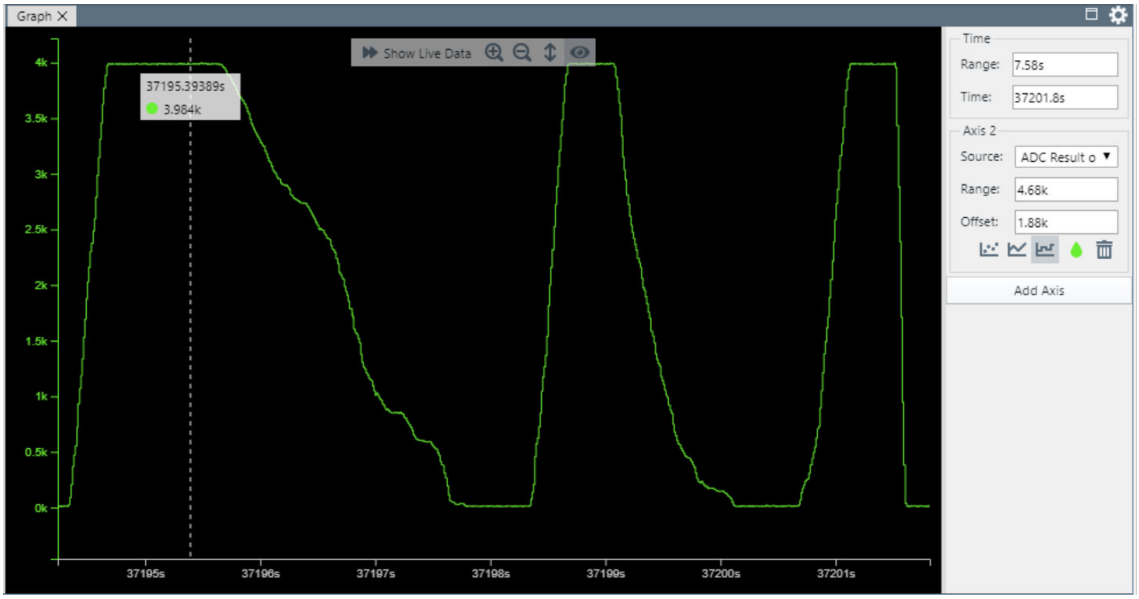
```
ADC0.MUXPOS = ADC_MUXPOS_AIN6_gc | ADC_VIA_PGA_gc; /*ADC channel AIN6->PA6, input
connected to the ADC via PGA*/
```

Info: The VIA bits in MUXPOS and MUXNEG are shared, so a value written to the VIA bit field in one of the two registers is updated in both. Therefore, it is impossible to have one input using the PGA and the other not using the PGA.

7. Verify that the solution builds with no errors by selecting *Build → Build Solution* from the top menu bar in Microchip Studio or pressing the F7 key.
8. Flash the device by selecting *Debug → Start without debugging* for the top menu bar in Microchip Studio or pressing the *Ctrl+Alt+F5* keys.
9. Press the force sensor and observe the plot in the MPLAB *Data Visualizer.*

Result: The result should look similar to Figure 7-4, where the top value is about 4k.

Figure 7-4. Assignment 6: Amplified Signal



8. Revision History

Doc. Rev.	Date	Comments
B	02/2021	PGABIASSEL setting updated to 50%
A	10/2020	Initial document release

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-7571-2

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820